Preprint
UCRL-CONF-204866

Performance-Oriented Privacy Preserving Data Integration

R. Pon, T. Critchlow

2nd International conference on Data Integration in the Life Sciences

July, 2005

Approved for public release; further dissemination unlimited

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

Performance-Oriented Privacy-Preserving Data Integration

Raymond K. Pon[1], Terence Critchlow[2]

[1] UCLA Computer Science Department,
Los Angeles, California, USA
pon@seas.ucla.edu
[2] Lawrence Livermore National Laboratory
Livermore, California, USA
critchlow@llnl.gov

Abstract. Current solutions to integrating private data with public data have provided useful privacy metrics, such as relative information gain, that can be used to evaluate alternative approaches. Unfortunately, they have not addressed critical performance issues, especially when the public database is very large. The use of hashes and noise yields better performance than existing techniques, while still making it difficult for unauthorized entities to distinguish which data items truly exist in the private database. As we show here, the uncertainty introduced by collisions caused by hashing and the injection of noise can be leveraged to perform a privacy-preserving relational join operation between a massive public table and a relatively smaller private one.

1 Introduction

Data is often generated or collected by multiple parties, and the need to integrate the resulting disparate data sources has been identified by the research community [1-6]. Although heterogeneity of the schemas is being addressed, most data integration approaches have not yet efficiently addressed privacy concerns.

Legal and social circumstances have made data privacy a significant issue [7-8], resulting in the need for Hippocratic databases (i.e., database that include privacy as a central concern) [9], particularly in sharing scientific or medical data. Without strong privacy guarantees, scientists often refuse to share data with others for reasons such as subject / patient confidentiality, proprietary / sensitive data restrictions, competition, and potential conflict and disagreement [10]. An application where both data sharing and privacy are important is biomedical research. In this domain research facilities frequently collaborate with each other, sharing experimental data and results. In particular, comparing genome sequences from different species has become an important tool for identifying functions of genes [11]. However, this necessitates integrating different databases. Unfortunately, while there is a significant amount of publicly available data, information provided by most companies, such as proprietary genome sequences, must be kept private.

More concretely, imagine that a scientist wishes to perform a query across a table in his private database (e.g., proprietary genome sequences) and a table in a public data warehouse (e.g., GenBank [12]) in the most efficient manner possible (shown in Figure 1). Ignoring privacy restrictions, the problem is reduced to a distributed database problem that can be solved by shipping the scientist's table to the warehouse and performing the join at the warehouse. However, if the scientist's data set is proprietary, it cannot be sent verbatim to the warehouse. The naive solution is for the scientist to download the entire public table to his local machine and perform the query there. But to do so would be prohibitively expensive if the public table is very large or the communications link is limited. It would be impossible if

the publicly available data cannot be duplicated, for example because of intellectual property constraints.

Assuming that all data sources are abstracted as relational tables and schema reconciliation has already been done, the problem can be formalized as the following: table $R = (A, B)$ from a small private database *db* is to be joined with table $S = (B, C)$ from a large data warehouse *dw* on column *B*, yielding the desired table *Goal* = *R*? $_{B}S$. Table *R* is private and the identity of the data items in *R* can not be known by any party other than the owner of *db*. Table *S* is publicly available and accessible. It is assumed that the system operates in a semi-honest model, where both parties will behave according to their prescribed role in any given protocol. However, there are no restrictions on the use of information that has been learned during the data exchange after the protocol is completed. Thus, from the privacy perspective, *dw* is treated as an adversary.

Our solution to this problem augments the well-known semi-join framework [13], "hiding" the actual values of the join column of table *R* by hashing them and including additional artificial values. The resulting collection is sent to the data warehouse to retrieve a subset of table *S* that includes the data required to answer the original query along with some false positives. Although, this method will not provide for absolute privacy (i.e., the adversary can infer something about the contents of table *R*), the hash / noise method can guarantee an upper bound on the amount of privacy loss when data is exchanged. By sacrificing a small amount of privacy, this method significantly reduces transmission costs compared to techniques that provide absolute privacy.

1.1 Challenges and Related Work

There are several challenges in privacy-preserving data integration, including: defining privacy; correctness; and efficiency. This section provides a short summary of the most relevant work being done by others to meet these challenges, as well as related work on general approaches to privacy preservation. Following this overview, Section 2 describes our privacy metric; Section 3 presents our hash / noise approach; Section 4 outlines a proof of concept implementation and initial experimental results, and; Section 5 summarizes our work and explores future roads of research.

First, a metric is needed to measure the amount of privacy loss that is incurred when data is exposed. In [14], variable privacy is proposed as a method in which some information can be revealed for some benefit. Privacy loss is likened to a communications channel, in which the difference between *a priori* (i.e., before data has been revealed) and *a posteriori* (i.e., after data has been revealed) distributions of information measures privacy loss. In [15], the likelihood of what can be inferred about a query posed by the user is used as a measure of privacy loss. In [16] and [17],

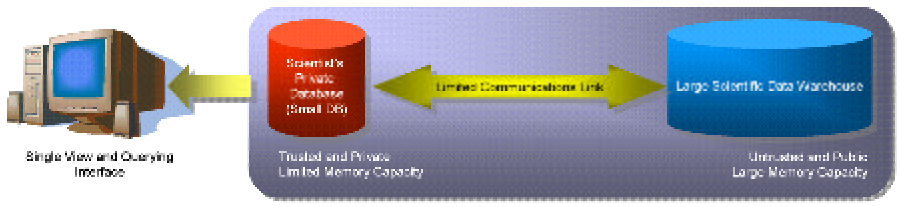


Fig. 1. General problem

a metric for measuring the inherent uncertainty of a random variable based on its differential entropy is used as a measure for privacy. These proposed metrics are related to relative information gain, which has also been used in many privacy-preserving applications [18], making it a likely candidate for measuring privacy loss.

The second challenge is producing exact and correct answers to queries posed by users. Work in privacy-preserving data mining [19-22] has focused on changing the actual values of data items so that the values of data items are hidden but the distribution of the perturbed data is similar to that of the original data distribution. However, the exact original data values can not be accurately recovered. While this is acceptable in data mining applications, the exact answers are required for data integration.

The third challenge is to perform the private join operation efficiently. It has been shown that to completely guarantee the privacy of the queries, the entire contents of table *S* should be downloaded [23]. However, in some cases this is not practical and an alternative solution is needed. If the user is willing to sacrifice a small portion of his data privacy, the join operation can be done without retrieving all of table *S*.

Commutative encryption-based approaches have been proposed to solve the private data integration problem [24-26]. These approaches take advantage of a family of encryption functions in which the order that data item are encrypted by two different keys does not matter. These techniques require the exchange of both parties' encrypted data so that they can both mutually encrypt each others' data, making them very expensive. Similarly, oblivious transfer [27-29] allows the user to secretly pose a query and only receive the result of the query and nothing else, but the encryption and transmission of all data items held by *dw* to the user is required.

There has also been work in private information retrieval schemes [23, 30], which allow a user to retrieve information from a database while maintaining the privacy of his query. In these schemes, table *S* would be replicated at multiple sites. Given a query, multiple queries are generated and sent to each of site such that no site can learn the actual original query by acting alone. However, users working with sensitive data would be unwilling to trust such a system if no guarantee of enforcement of non-collusion among the sites.

Our hash / noise method takes an approach similar that to the one discussed in [15], which takes advantage of collisions caused by hashes to introduce uncertainty in the true contents of a private database's table. A hash value is generated for each data item in both tables each time a query is posed. The size of the hash is varied to control the amount of privacy loss, so traditional indexing mechanisms cannot be used to accelerate querying time. A sequential scan of both tables involved in the join is necessary to compute the hash values of all data items in both tables. As a result, the join operation becomes a very expensive operation. There has been work in using Bloom filters to make joins in a distributed database system more efficient and private [31-33]. Similar to the hashing approach, however, Bloom filters would require a sequential scan of both tables to apply a Bloom filter to each of the data items and would not allow the use of traditional indexing mechanism to speed up querying.

In contrast, to these two approaches, our hash/noise method approach uses a set of fixed hashing and artificial hash values (i.e., noise) to control the amount of uncertainty in the identity of the join column values, thereby controlling the level of privacy loss incurred. Because the hashes are known in advance, we can store and index the resulting hash values in the database and would not need to recompute them for each query, enabling indexes to be used to speed up querying. Because

the hash functions are known in advance, a dictionary-attack is possible but is partially alleviated by using artificial hash values.

Furthermore, privacy control by hash truncation alone as suggested by [15] is very coarse. For example, suppose that a 16-bit hash does not satisfy a given privacy constraint, so a 15-bit hash was selected instead. However, the 15-bit hash doubles the collision rate of the 16-bit hash, doubling the size of the candidate set for the join result. In contrast, the same 16-bit hash with additional artificial hash values could have satisfied the same privacy constraint and yield fewer records in the candidate set.

2 Privacy Metric

For our work, we use relative information gain [34] as a basis for a metric to measure privacy loss when data is exchanged. Relative information gain is closely related to entropy, which is the amount of uncertainty in a random variable X. If the random variable X can take on a set of finite values $x_1, x_2, \ldots x_n$, then its entropy is defined as:

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log_2 P(X = x_i) \quad (1)$$

The conditional entropy $H(X \mid Y)$ is the amount of uncertainty in X after Y has been observed. Relative information gain, or the fraction of information revealed by Y about X, is defined as:

$$RIG(X;Y) = \frac{H(X) - H(X \mid Y)}{H(X)} \quad (2)$$

Privacy loss can be thought as the amount of information gained by an adversary about the contents of set of sensitive data items, which in this case are the contents of column B of table R. If *dw* (i.e., the adversary) has no knowledge about the distribution of column B of table R, then it can only assume that each value that belongs to the domain of B (i.e., U) are equally likely to occur. Let $\tilde{R}$ be a random variable describing the column B values (the only information revealed in a semi-join by *db*), of a tuple in table R. Absolute privacy loss p_{abs} is defined as the relative information gain on $\tilde{R}$ when any data set N is revealed to *dw* by *db*. By doing a simple substitution with equation 2, absolute privacy loss is:

$$p_{abs} = \frac{H(\tilde{R}) - H(\tilde{R} \mid N)}{H(\tilde{R})} = \frac{\log_2 |U| - H(\tilde{R} \mid N)}{\log_2 |U|} \quad (3)$$

It is possible that an adversary will make use of any available information to infer the contents of table R, in particular the contents of table S, since it is publicly available. Thus, relative privacy loss is defined as:

$$p_{rel} = \frac{H(\tilde{R} \mid S) - H(\tilde{R} \mid N)}{H(\tilde{R} \mid S)} \quad (4)$$

In this case, the adversary uses the distribution of values in column B of table S as a hint to the possible distribution of values in column B of table R. $H(\tilde{R} \mid S)$ (the uncertainty of the join column values of a tuple in table R given the contents of table S) can be found by directly applying equation 2 on the distribution of values in column B of table S. Because this metric captures the information gained by an adversary with respect to its current knowledge in contrast to absolute privacy loss, it is the metric we have chosen for evaluation of our approach.

3 Privacy-Preserving Distributed Join

Figure 2 outlines our approach to finding $R?_B S$ when a privacy constraint exists. The first step projects column B from table R and applies a hashing function h to each value in column B, yielding table $h(R)$ with column $h(B)$. Step 2 generates artificial hash values, yielding table n. In step 3, table N is derived from the union of n and $h(R)$. Table N is then shipped to the data warehouse in step 4. At the data warehouse in step 5, table S and N are joined on column $h(B)$, yielding table F. Table F is a set of tuples from dw that contain the final result of the join operation and which is shipped to db in step 6. The final result, $Goal$, is found by filtering out the false positives in F by joining tables R and F.

3.1 Privacy Constraint Satisfaction

Different hash functions yield different collision rates. Hash functions with large ranges tend to yield low collision rates; whereas, hash functions with smaller ranges tend to yield high collision rates. A hash function h with a high collision rate introduces large amounts of uncertainty about x when $h(x)$ is known. This uncertainty is used to mask the true identity of a join column value in table R. Hash functions also hide clusters of data by hashing clustered values to uniformly-distributed hashed values. A hash function with a high collision rate has the side effect of "compressing" the values of column B from table R since a single hash value can be used to represent multiple actual values. However, if the collision rate is too high, many false positives will occur in F due to the high number of collisions, yielding unnecessary transmission costs. Thus, it is important to use a hashing function that provides an acceptable level of performance while providing enough uncertainty to meet the privacy constraint.

It is computationally expensive to dynamically compute the hash values resulting from a new hash function with a different size each time a query is posed on a large data warehouse table. Furthermore, dynamic generation of values prevents indexing mechanism from being used to during the join operation in step 5. Our approach is to predefine a set of m hash functions $h_1,h_2,\ldots,h_m$ of different sizes. The result of each of these hash functions to column B on table S are stored explicitly (in m different columns) and indexed.

When the user performs a join on his private table R and the public table S, the privacy loss incurred with respect to the contents of table S is constrained to not exceed p_{rel}. In other words:

$$p_{rel} \geq \frac{H(\tilde{R} \mid S) - H(\tilde{R} \mid N)}{H(\tilde{R} \mid S)} \qquad (5)$$

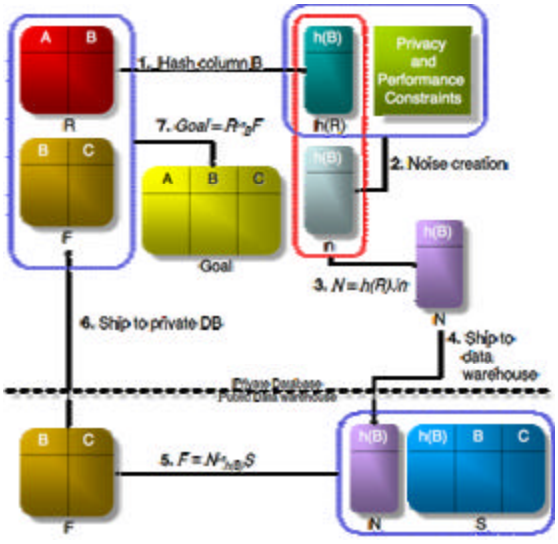


Fig. 2. Privacy-preserving distributed join

Assuming a uniformly-distributing hash function, the number of real values that hash to the same hash value is estimated to be $\frac{|U|}{|H|}$, where $|U|$ is the size of the domain of possible values for column B (the universe) and $|H|$ is the range size of hash function h. H is the set of possible values in the range of h. For any given hash value, $\frac{|U|}{|H|}$ possible values could have been used as input into the hash function and could have belonged to table R. For a set of $|N|$ hash values, there is a total of $|N|\frac{|U|}{|H|}$ possible values that data items in column B of table R can take on with equal probability. Thus, $H(\tilde{R} \mid N)$ is estimated as:

$$H(\tilde{R} \mid N) = \log_2\left(\mid N \mid \frac{\mid U \mid}{\mid H \mid} \right) \quad (6)$$

By combining equations 5 and 6, the constraint on $|N|$ for a given p_{rel} is found to be:

$$|N| \geq \frac{|H|}{|U|} 2^{(1-p_{rel})H(\tilde{R}|\tilde{S})} \quad (7)$$

Applying equation 7 to each hash function, the minimum number of hash values $|r_1|, |r_2|, \ldots, |r_m|$ for all m available hash functions on dw can be found.
We can estimate the number of unique hash values generated by hashing each tuple in R with h_i analogously to [15] as:

$$|h_i(R)|_{est} = \left(1 - \left(1 - \frac{1}{|H_i|} \right)^{|R|} \right) |H_i| \quad (8)$$

Then the actual size of the hash value set N_i that db would send to dw, if hash function h_i is selected, is:

$$|N_i| = \max(r_i, |h_i(R)|_{est}) \quad (9)$$

Note that $|N_i| = |h_i(R)|$, so it may be necessary to add artificial hash values to the set N sent by db to dw in addition to $h_i(R)$. This can be done by randomly selecting $|N_i| - |h_i(R)|$ hash values that belong to the range of h_i. The set of artificial hash values is denoted as n_i, where $N_i = h_i(R) \cup n_i$.

3.2 Cost Estimation

To select the appropriate hash function for the data exchange, the transmission cost normalized with respect to the brute-force method (i.e., downloading table S from dw to db) $cost_i$ can be estimated. It is assumed that transmissions costs will dominate the execution costs of the overall join operation since the system will be operating over a limited communications link and search time is kept low with the use of indices.

If the brute-force method was used, $c_t|S|$ time units are required to transmit $|S|$ records from dw to db where c_t is the cost associated with transmitting a single record returned by dw in bytes. The cost of the hash / noise method can be estimated to be the sum of the cost of transmitting hash values from db to dw and the cost of transmitting the set of candidate tuples F returned by dw to db. The cost of sending the hash values is $c_h|N_i|$ time units for a hash function h_i, where c_h is the cost associated with transmitting a single hash value. The cost of the tuples returned by dw to db after the hash values have been sent is $c_t|F|$. Thus, the transmission cost normalized with respect to the brute-force method is summarized as:

$$cost_i = \frac{c_h|N_i| + c_t|F|}{c_t|S|} = \frac{|F|}{|S|} + \frac{c_h|N_i|}{c_t|S|} \quad (10)$$

Equation 10 shows that as the cost-ratio c_h/c_t approaches zero, the cost of sending hash values $\frac{c_h|N_i|}{c_t|S|}$ becomes small. In other words, as the size of the tuples returned increases the cost of sending the hash becomes insignificant. As $|F|$ approaches $|S|$, the performance of the hash / noise method is similar to that of the brute-force method; whereas, when $|F| << |S|$,

we see significant performance improvement over the brute method. While $|F|$ is not known until the query has been executed, it can be estimated to be the average number of tuples returned by *dw* given the characteristics of the hash function and the contents of *dw*. It is found that on average for a given hash value, the number of values in column *B* that will collide to the some hash value is $\frac{|S|}{|H_i|}$ for a hash function h_i. Consequently, the average number of tuples returned by *dw* to *db* is $\frac{|S|}{|H_i|}|N_i|$. Thus the normalized transmission cost $cost_i$ for a hash function h_i is estimated to be:

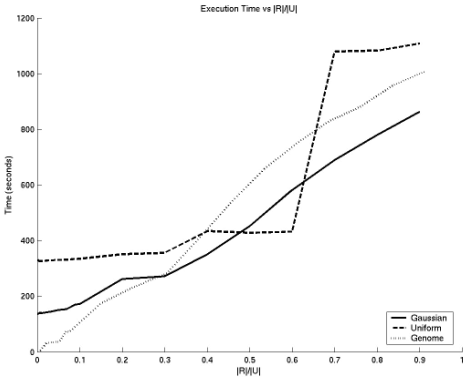


Fig. 3. Execution times for variable $|R|/|U|$. Target p_{rel} = 0.01 and c_t/c_r = ½

$$cost_i = \frac{c_h|N_i| + c_t\frac{|S|}{|H_i|}|N_i|}{c_t|S|} \qquad (11)$$

The hash function h_i, with the appropriate N_i found with equation 9 that yields the lowest normalized transmission cost according to equation 11 is selected as the hash function for the data exchange. Clearly, if $cost_i$ = 1, it would be more advantageous to download *S* since the cost of doing so is either less than or equal to the cost of our hash-noise approach without any loss of privacy.

4. Implementation and Results

A prototype of this system was implemented in Java using MySQL [35]. Borrowing a technique from [15], eight hash functions were created by simply truncating the result of the MD5 hash [36]. Eight sets of hash values were generated for each *B* column value by truncating the result of the MD5 hash of a column *B* value to various bit sizes ranging from 8 to 16 bits. The hash value sets were stored and indexed in *dw* along with their respective *S* table. $H(\tilde{R}|S)$ was computed offline and stored for each *S* table.

Three sets of data were used for three instances of table *S*. The first two were each comprised of 2.5 million synthetically generated tuples. The values of column *B* for table *S* were generated with a uniform distribution of values from 0 to 99,999 for the first set. The second set's column B values were generated with a Gaussian distribution of values from 0 to 99,999 with a mean of 50,000 and a standard deviation of 1000. The third set of data was the "alignment block in rat chain of chromosome 10" table, taken from the UCSC Genome Browser Project [37]. The genome data set contains approximately 2.4 million records and was biased towards low join column values.

The size of the domain *U* for the uniformly and Gaussian-distributed join column values was 100,000. There were approximately 123,598 different values for the join column in the genome data set, so the size of domain *U* for join column values was approximated to be 2^{17}. Unless

otherwise specified, the cost-ratio c_h/c_t was ½ (i.e., the cost of transmitting of a hash value is half the cost of transmitting a record from table *S*).

For each experiment, the *R* tables were generated randomly. The *R* tables to be joined with a uniformly or a Gaussian-distributed table *S* were generated by randomly selecting a value for column *B* from the range of 0 to 99,999. The *R* tables to be joined with the genome data were generated by randomly selecting tuples from the "summary information about chain of rat" table (also available from [37]). For each data point plotted, five *R* tables were randomly generated, each of which was joined with table *S* using the hash / noise method fives times. The maximum and minimum observed values of each studied parameter were ignored, and the rest were averaged. Timings were taken using a dual processor 1.3 GHz Dell workstation.

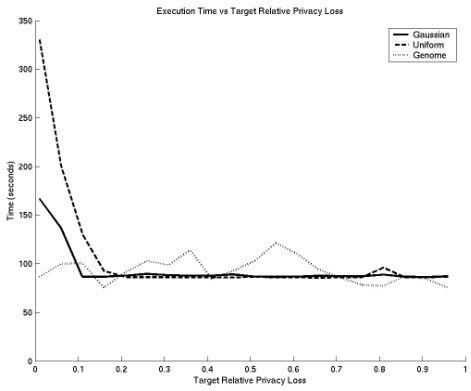


Fig. 5. Execution times for variable target p_{rel}. $|R|/|U|$ = 0.1 and c_h/c_t = ½

4.1 Performance Analysis

In this section, we will study the performance implications (i.e., the size of data sets transmitted and execution time) of different distributions of the private table and different privacy requirements.

4.1.1 Effect of Private Table Distributions

To begin the execution time analysis, we study the effect of different private table distributions on performance by varying the size of table *R* in relation to the size of the set of possible key values *U* ($|R|/|U|$) and fixing the required relative privacy loss to not exceed 0.01. Figure 3 shows how execution time varies as $|R|/|U|$ changes. Figure 4 shows how the size of the transmitted sets $|N|$ and $|F|$ varies as $|R|/|U|$ changes. For each of the execution time tests, the transmission cost of transmitting a hash value was equivalent to transmitting a 4-byte integer, and the cost of transmitting a tuple from *S* was equivalent to transmitting two 4-byte integers. For a Gaussian distribution and genome data distributions of table *S*,

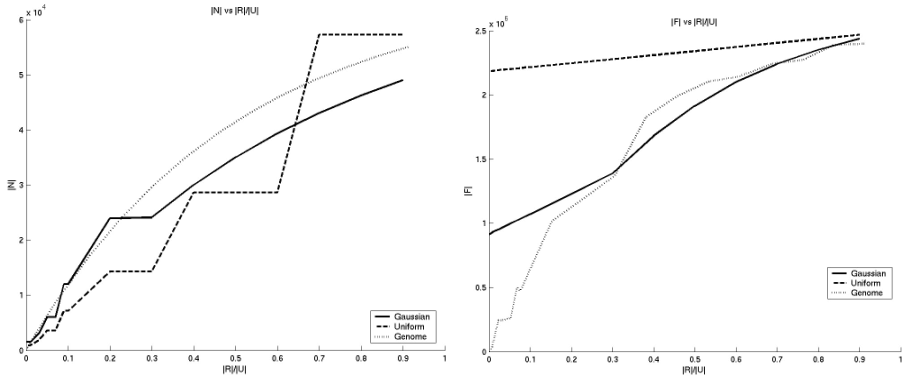


Fig. 4. Set sizes $|N|$ and $|F|$ for variable |R|/|U|. Target p_{rel} = 0.01 and c_h/c_t = ½

execution time increases linearly as $|R|/|U|$ increases as do the sizes of N and F. Thus, as expected, the processing (i.e., transmission and computation time) of the two intermediate sets dominate the execution time for these two data distributions.

For a uniform distribution of table S, the execution time behaves as a step function, transitioning when $|R|/|U| = 0.6$. Figure 4 shows that $|N|$ increases along with the execution time curve; whereas, $|F|$ remains relatively constant. While initially surprising, as shown in Figure 8, when $|R|/|U|$ transitions from 0.6 to 0.7, the system experiences the largest increase in hash size $|H|$, resulting in far fewer collisions; and, consequently many more hash values are sent to *dw* to meet the privacy constraint. Because, the largest hash size $|R|/|U|$ values increase occurs at much lower distributions, any for the Gaussian and genome less apparent for sharp increases in execution times are those distributions.

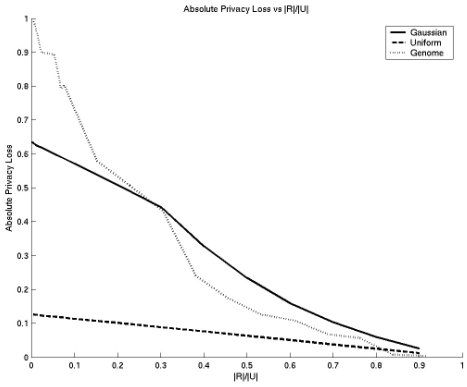
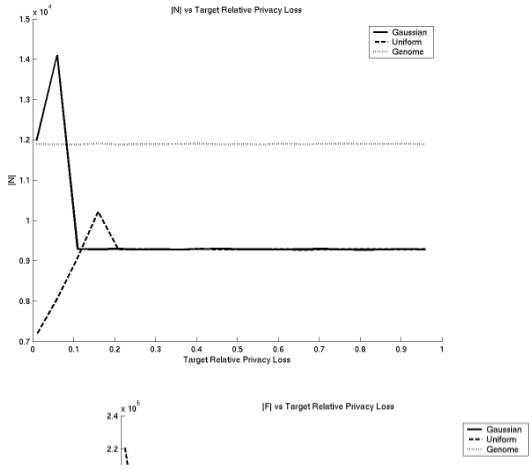


Fig. 7. Varying absolute privacy. Target p_{rel} = 0.01 and c_h/c_t = ½

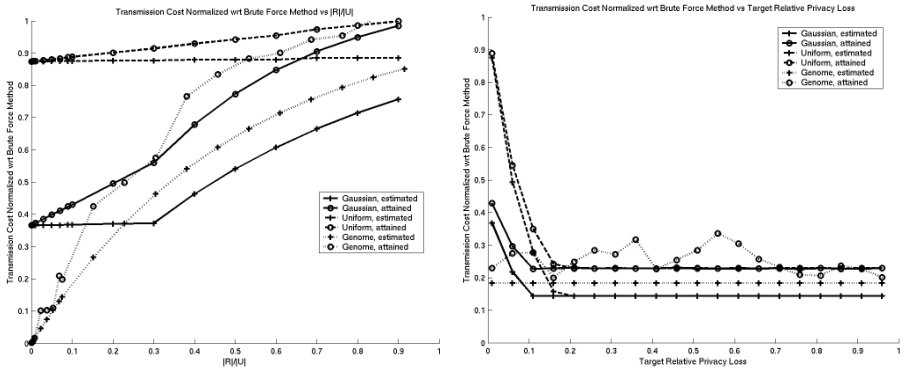
Comparing the behavior of the various distributions, the execution time of the distributed join operation is directly related to the size of tables R, N, and F for the Gaussian data distribution and the genome data distribution. However, for a uniform distribution, the execution time is generally independent of $|R|/|U|$, except when there is a large transition in hash values used, because the transmission of noise and false-positives dominate the cost. From this figure, it can also be seen that the execution times for join operations operating over the genome data distribution are lower than for the Gaussian distribution, which are usually lower than for the uniform distribution. Less uniform distributions will usually result in better execution times because they are more biased and thus will have less entropy. Uniform distributions have the most entropy of any distribution, requiring either far more hash values or far more false positives to be returned by *dw* to satisfy the privacy constraint.



4.1.2 Effect of Privacy Requirements

In the second set of execution time analyses, we will study performance implications of different privacy requirements by fixing $|R/|U|$ to 0.1 and by varying the maximum privacy loss, or the target relative privacy loss p_{rel}, from 0.01 to 0.96, in intervals of 0.05. Figure 5 shows how execution times vary as the target p_{rel} changes. Figure 6

shows how $|N|$ and $|F|$ vary as the target p_{rel} changes in the second graph. Intuitively, as the privacy constraint is relaxed, execution times for both the Gaussian and uniform data distributions decrease since fewer hash values are needed to satisfy the privacy constraint. For any join operation whose target p_{rel} is greater than 0.21, the execution times, $|N|$, and $|F|$ remain constant. In such cases, $|h(R)|$ is large enough to satisfy the privacy constraint without any noise. Thus, there is very little performance gain by increasing the target relative privacy loss greater than 21% for private tables containing only 10% of the total possible keys.



(a) **(b)**

Fig. 8. Varying normalized transmission costs with respect to the brute-force method. (a) Target p_{rel} = 0.01 and σ/μ = ¼. (b) $|R|/|U|$ = 0.1 and σ/μ = ¼

Figure 5 also shows that the execution time of the genome data set remains relatively constant, with minor variations in execution times due to the randomness of data items in set *R* and consequently the high randomness of data items in set *F*. Furthermore, $|N|$ remains constant regardless of the target privacy; and consequently, only the varying sizes of table *F* contribute to the variation in execution times, which is determined by the random selection of tuples in table *R*. This is shown in the second graph of Figure 6. The variance in execution times is more than that of the other distributions because the data in the genome data set is much less uniformly distributed than the other two distributions.

In summary, when target p_{rel} is low, there is more variation in execution times for the Gaussian and uniform distributions. When the privacy constraint is relaxed, there is little or no change in execution times.

4.2 Absolute Privacy Loss Analysis

Figure 7 shows how absolute privacy loss varies as $|R|$ changes and the target p_{rel} is fixed at 0.01. For the uniform distribution, the absolute privacy loss is kept very low and close to the target p_{rel} of 0.01 since satisfying the relative privacy loss constraint for a uniform distribution is almost identical to satisfying an absolute privacy constraint of the same magnitude. However, for the Gaussian and genome data distributions, the absolute privacy loss differs greatly from the target relative p_{rel}, because far less effort is required to satisfy the relative privacy loss constraint than that required to satisfy an absolute privacy loss constraint of equal magnitude due to less uniformity in these distributions. For non-uniform distributions, achieving low absolute privacy loss would be much more expensive than achieving low relative absolute privacy loss; whereas, the cost for achieving both for a uniform distribution would be relatively the same.

Figure 7 also shows that as $|R|/|U|$ increases, absolute privacy loss decreases. In general, as $|R|/|U|$ increases, the data revealed by *db* to *dw* increases. As a result, the pool of possible values that an adversary can use to infer the actual values of column *B* in table *R* increases as well, resulting in far greater uncertainty about the actual value of a column *B* value in table *R*.

4.3 Hash Selection Analysis

In this analysis, we determined the size of the selected hash function that yields the lowest transmission cost increases as $|R|/|U|$ increases, for all distributions. We experimented with hash sizes ranging from 8 to 16 bits because any larger hash sizes, such as 17 bits would yield almost no collisions. It was found that as the uniformity of table *S* increases, a wider range of hash values is required to account for any variations in sizes of table *R* provided by a user. Depending on the size of $|R|/|U|$, for the uniform distribution, hash sizes ranging from 10-bits to 16-bits are required. For the Gaussian distribution, hash sizes ranging from 12-bits to 16-bits are required. Finally, for the genome data set, hash sizes ranging from 14-bits to 16-bits are needed.

4.4 Transmission Cost Analysis

In this set of analyses, the transmission costs of the hash/noise method in relation to the brute-force are studied. The observed normalized transmission cost based on equation 10 using the observed $|F|$ is compared to the estimated normalized transmission cost based on equation 11. The first graph of Figure 8 shows that the hash / noise method works well when $|R|/|U|$ is very low, and especially well when the distribution of key values in table *S* is very biased. For uniform distributions of table *S* and a target p_{rel} of 0.01, the transmission costs of the hash / noise method was 90% or more of the transmission costs of the brute-force method, costing as much as the brute-force method. For a Gaussian-distributed data set, the transmission costs ranged from 35% to 95% of the brute-force method, depending on $|R|/|U|$. For the skewed genome data set, the transmission cost also varied significantly depending on the size of $|R|/|U|$. The second graph shows that the transmission cost steeply decreases as the target p_{rel} increases from 0.01 to 0.2 for both Gaussian and uniform distributions. For any target p_{rel} greater than 0.2, transmission costs are 25% of that of the brute-force method, for all distributions. The general behavior of steeply decreasing and flattening out was predicted by the estimated normalized transmission cost curves, but the actual transmission costs were not accurately estimated. For the less uniform genome data, the transmission costs remain relatively constant with an average of 25% of that of the brute-force method, for all target relative p_{rel} values and when $|R|/|U|$ is 0.1. Like for the other distributions, the general behavior of the observed transmission cost curve was predicted by the estimated transmission cost curves, but the actual transmission costs were poorly predicted.

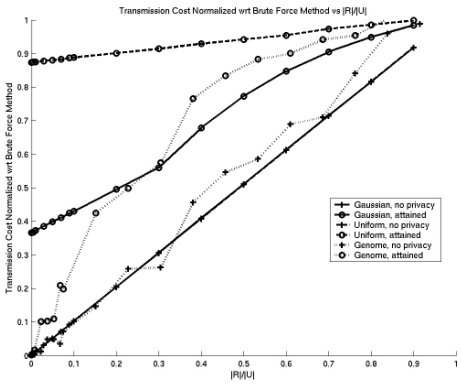


Fig. 9. Attained normalized transmission costs of join with privacy constraints and join without privacy constraints. Target p_{rel} = 0.01 and c_h/c_t = ½. Cost of transmitting the key of a record from *db* is half the cost of transmitting a tuple from *dw*

Figure 9 compares the attained normalized transmission cost of the hash / noise method with the cost of simple semi-joins (i.e., no privacy constraints enforced). The graph shows that $|R|/|U|$ is directly proportional to what the cost of the semi-join would be. The graph summarizes how much more the hash / noise method costs to satisfy a maximum relative privacy loss of 0.01 in comparison to a semi-join, which provides for no privacy. Using the hash / noise method, it is very expensive to achieve a maximum relative privacy loss of 0.01 when the distribution of the column *B* values of the *S* table is uniform. In contrast, when the *S*

table is non-uniform, there is much less additional cost for the added privacy that the hash/noise method provides.

5. Conclusion and Future Work

A practical solution to the private date integration problem must maintain privacy while remaining efficient. Based on the metric of relative information gain, we have presented an efficient approach to performing joins between a relatively small database and a large, public data repository. By making use of predefined hash functions and noise injection to satisfy the privacy constraints, traditional indexing mechanisms can be used. Thus the total cost of a distributed join is dominated by transmission costs rather than by search and computational costs.

Based on our preliminary results, several future research directions can be pursued.

Our current cost estimation uses the average number of collisions to estimate the number of tuples to be returned by *dw*, which works well for uniformly-distributed data but poorly for non-uniformly distributed data. In future work, additional features such as the distribution of table *S* could be incorporated into the estimate. This work can also be expanded to infinite domains (e.g., people's names), specifically to develop a privacy loss metric relevant to these domains. Additionally, our method only protects the privacy of data over a single query; and, it may be possible for adversaries to make inferences over multiple queries. Perhaps, some caching can be used to avoid exposing the same private data set more than once. Finally, the presented hash/noise technique only works for the equijoin operation. There may be a need to develop methods to protect the privacy of data that are processed by general joins.

Our promising initial results show the merit of using hashing and noise injection to solve the problem of efficiently integrating small amounts private data with large amounts of public data. In comparison to other related approaches, the hash/noise technique does not assume non-collusion, does not require downloading the entire data warehouse table, leverages existing indexing mechanisms, and provides for finer-grain control of privacy than simple hashing.

7. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48. The authors would like to thank Tina Eliassi-Rad, David Buttler, and Roderick Son for their valuable input.

References

1. S. Phillippi and J. Kohler, "Using XML Technology for the Ontology-Based Semantic Integration of Life Science Databases," IEEE Transactions on Information Technology in Biomedicine, vol. 8, no. 2, pp. 154-160, June 2004.
2. Tomasic, L. Raschid, and P. Valduriez, "Scaling Access to Heterogeneous Data Sources with DISCO," IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 5, pp. 808-823, Sept/Oct 1998.
3. S.B. Davidson, et al, "Transforming and Integrating Biomedical Data using Kleisli: A Perspective," ACM SIGBIO Newsletter, vol. 19, no. 2, pp. 8-13, 1999.
4. Z. Lacroix, O. Boucelma, and M. Essid, "The Biological Integration System," in Proceedings of WIDM '03, pp. 45-49, New Orleans, LA, Nov. 7-8, 2003.
5. M. Alvarez, et al, "FINDER: A Mediator System for Structured and Semi-Structured Data Integration," in Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA '02), pp. 847, Aix-en-Provence, France, Sept. 2-6, 2002.
6. L.M. Haas, et al "DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources," IBM Systems Journal, vol. 40, no. 2, pp. 489-511, 2001.
7. B. Thuraisingham, "Data Mining, National Security, Privacy and Civil Liberties," ACM Special Interest Group on Knowledge

Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter, vol. 4, no. 2, pp. 1-5, June 2002.
8. M.S. Olivier, “Database Privacy: Balancing Confidentiality, Integrity and Availability,” ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter, vol. 4, no. 2, pp. 20-27, June 2002.
9. R. Agrawal et al, “ Hippocratic Databases,” in Proceedings of the 28th Very Large Databases (VLDB) Conference, Hong Kong, China, 2002.
10. T.D. Sterling and J.J. Weinkam, “Sharing Scientific Data,” Communications of the ACM, vol. 33, no. 8, pp. 113-119, Aug. 1990.
11. F. S. Collins, E. D. Green, A. E. Guttmacher, and M. S. Guyer, "A vision for the future of genomics research," Nature, vol. 422, no. 6934, pp. 835-847, 2003.
12. NCBI, "GenBank," [Online] Available: http://www.ncbi.nlm.nih.gov/Genbank/index.html, 2004.
13. P.A. Bernstein and D.W. Chiu, “Using Semi-Joins to Solve Relational Queries,” Journal of the ACM, vol. 28, no. 1, pp. 25-40, Jan. 1981.
14. P.L. Vora, “Towards a Theory of Variable Privacy,” in review, May 7, 2003.
15. G. Schadow, S.J. Grannis, and C.J. McDonald, “Privacy-Preserving Distributed Queries for a Clinical Case Research Network,” in Proceedings of IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining, Maebashi City, Japan, 2002.
16. D. Agrawal and C.C. Aggarwal, “On the Design and Quantification of Privacy Preserving Data Mining Algorithms,” in Proceedings of Principles of Database Systems (PODS) 2001, pp. 247-255, Santa Barbara, CA, 2001.
17. C. Clifton, M. Kantarcioglu, and J. Vaidya, “Defining Privacy for Data Mining,” in Proceedings of the National Science Foundation Workshop on Next Generation Data Mining, Nov. 1-3, 2002, Baltimore, MD.
18. C. Clifton, et al, “Privacy-Preserving Data Integration and Sharing,” in Proceedings of Data Mining and Knowledge Discovery (DMKD) ’04, Paris, France, June 13, 2004.
19. J. Vaidya and C. Clifton, “Privacy Preserving Association Rule Mining in Vertically Partitioned Data,” in Proceedings of ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) International Conference on Knowledge Discovery and Data Mining (KDD ’02), Edmonton, Alberta, Canada, 2002.
20. S. Agrawal, V. Krishnan, and J. Haritsa, “On Addressing Efficiency Concerns in Privacy-Preserving Data Mining,” in Proceedings of the International Conference on Database Systems for Advanced Applications (DAFSAA) 2004, pp. 113-114, Jeju Island, Korea, Mar. 17-19, 2004.
21. W. Du and Z. Zhan, “Using Randomized Response Techniques for Privacy-Preserving Data Mining,” in Proceedings of ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) International Conference on Knowledge Discovery and Data Mining (KDD ’03), Aug. 24-27, 2003.
22. R. Agrawal and R. Srikant, “Privacy-Preserving Data Mining,” in Proceedings of the 2000 ACM International Conference on Management
of Data, pp. 439-450, Dallas, TX, 2000.
23. B. Chor et al, “Private Information Retrieval,” Journal of the ACM, pp. 965-982, vol. 45, no. 6, Nov. 1998.
24. R. Agrawal, A. Evfimievski, and R. Srikant, “Information Sharing Across Private Databases,” in Proceedings of the Special Interest Group on Management of Data (SIGMOD) 2003, pp. 86-97, San Diego, CA, June 9-12, 2003.
25. M. Kantarcioglu and C. Clifton, “Assuring Privacy when Big Brother is Watching,” in Proceedings of Data Mining and Knowledge Discovery (DMKD) ’03, San Diego, CA, June 13, 2004.
26. C. Clifton, et al, “Tools for Privacy Preserving Distributed Data Mining, ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD) Explorations Newsletter, vol. 4, no 2, pp. 28-34, Dec. 2002.
27. M. Naor and B. Pinkas, “Efficient Oblivious Transfer Protocols,” in Proceedings of Society of Industrial and Applied Mathematics (SIAM) Symposium on Discrete Algorithms, Washington, DC, Jan. 7-9, 2001.
28. M. Bellare and S. Micali, “Non-Interactive Oblivious Transfer and Applications,” in Proceedings on Advances in Cryptology, pp. 547-557, Santa Barbara, CA, 1989.
29. M.J. Freedman, K. Nissim, and B. Pinkas, “Efficient Private Matching and Set Intersection,” in Proceedings of Eurocrpyt 2004, Interlaken, Switzerland, May 2-6, 2004.
30. Y. Gertner et al, “Protecting Data Privacy in Private Information Retrieval Schemes,” in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pp. 151-160, Dallas, TX, 1998.
31. J.K. Mullin, “Optimal Semijoins for Distributed Database Systems,” IEEE Transactions on Software Engineering, vol. 16, no. 5, pp. 558-560, May 1990.
32. J.M. Morrissey and W.K. Osborn, “Distributed Query Optimization Using Reduction Filters,” in Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, vol. 2, pp. 707-710, May 24-28 1998.
33. S. Bellovin and W. R. Cheswick, “Privacy-Enhanced Searches Using Encrypted Bloom Filters,” in Proceedings of DIMACS/Portia Workshop on Privacy-Preserving Data Mining, Piscataway, NJ, Mar. 15-16, 2004.
34. C.E. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July

and Oct. 1948.
35. MySQL AB, “MySQL: The World’s Most Popular Open Source Database,” Aug. 2004; http://dev.mysql.com/
36. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1997, pp. 347.
37. UCSC Genome Bioinformatics, “UCSC Genome Browser Home,” Aug. 2004; http://genome.ucsc.edu/

University of California
Lawrence Livermore National Laboratory
Technical Information Department
Livermore, CA 94551